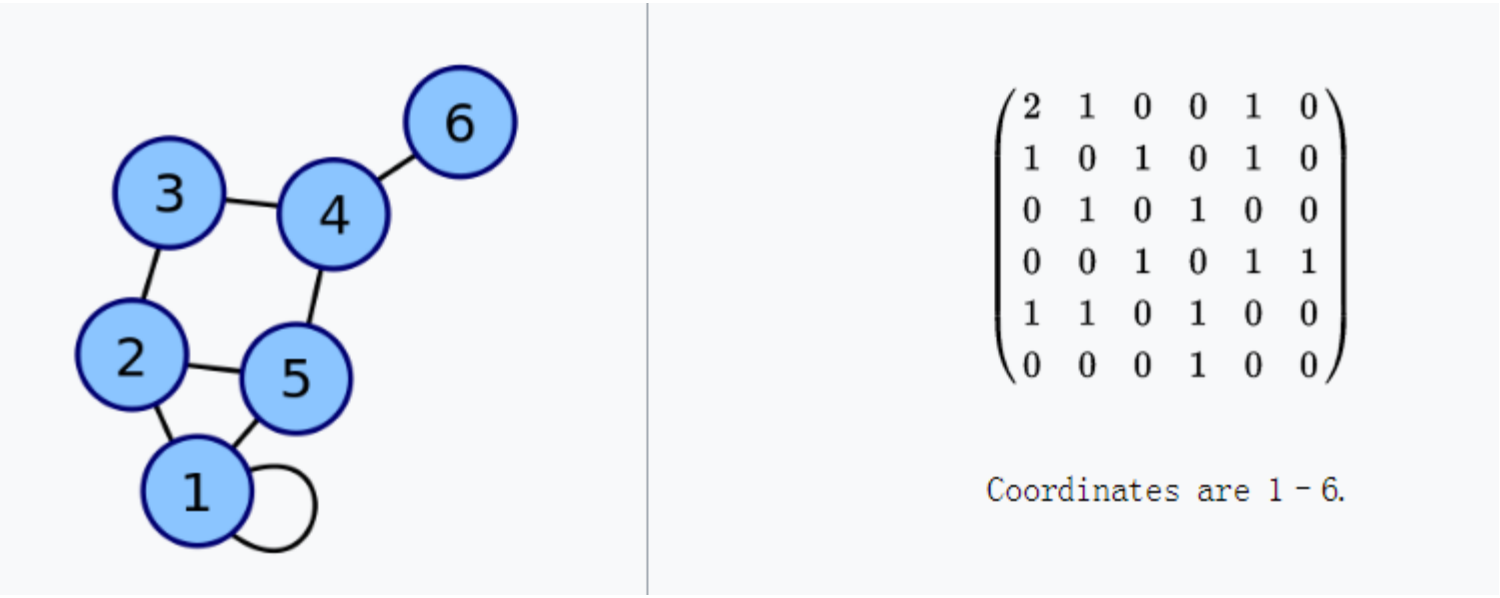
Part I: Network Embedding: Recent Progress and Applications

Xin Zhao (@赵鑫RUC)
batmanfly@qq.com
Renmin University of China

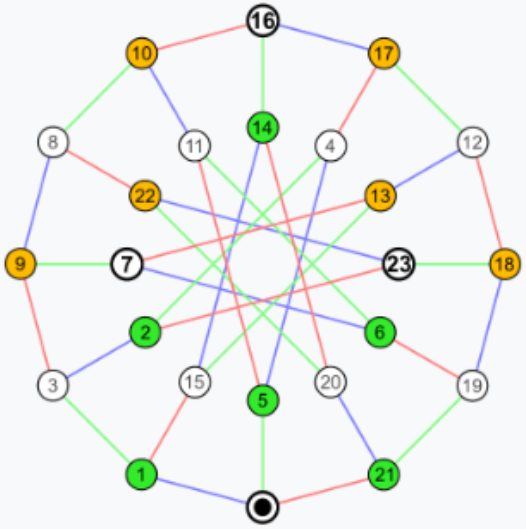
Traditional Network Representation

- How to represent a graph

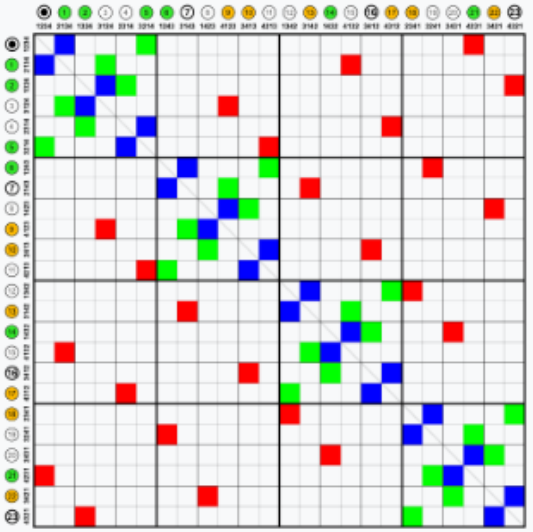


Traditional Network Representation

- How to represent a graph



Nauru graph



Coordinates are 0 - 23.

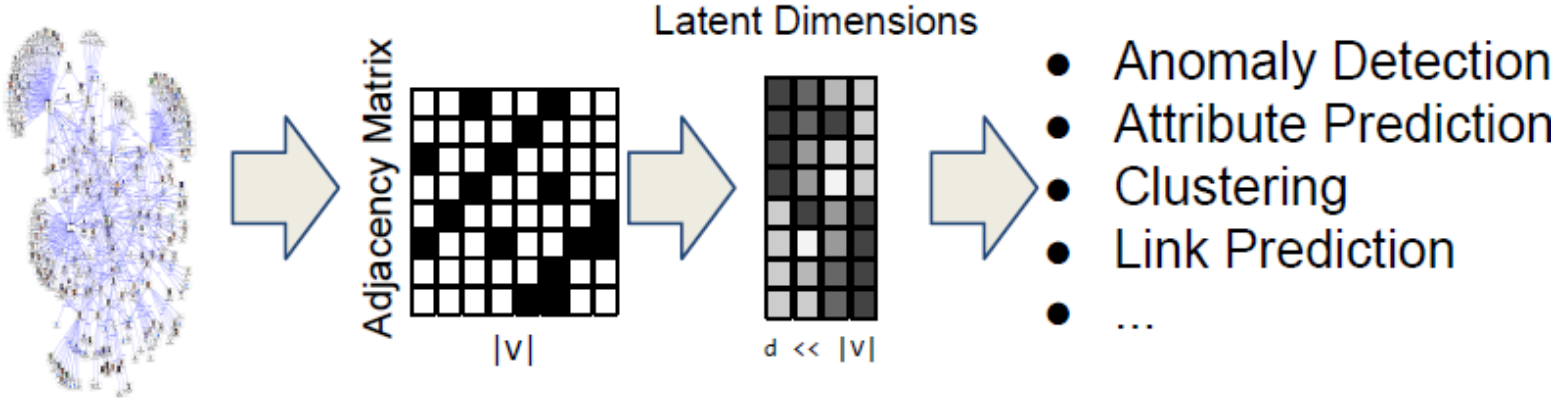
White fields are zeros, colored fields are ones.

Concepts

- Representation learning
  - Using machine learning techniques to derive data representation
- Distributed representation
  - Different from one-hot representation, it uses dense low-dimensional vectors to represent data points
- Embedding
  - Mapping information entities into a low-dimensional space

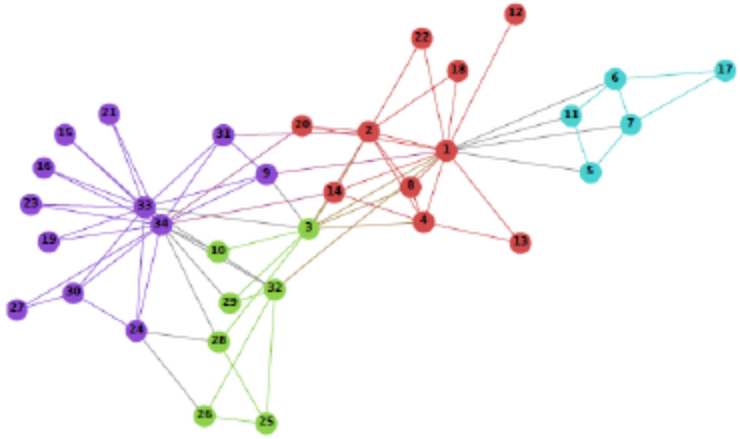
What is network embedding?

- Map the nodes in a network into a low-dimensional space
  - Distributed representation for nodes
  - Similarity between nodes indicate the link strength
  - Encode network information and generate node representation

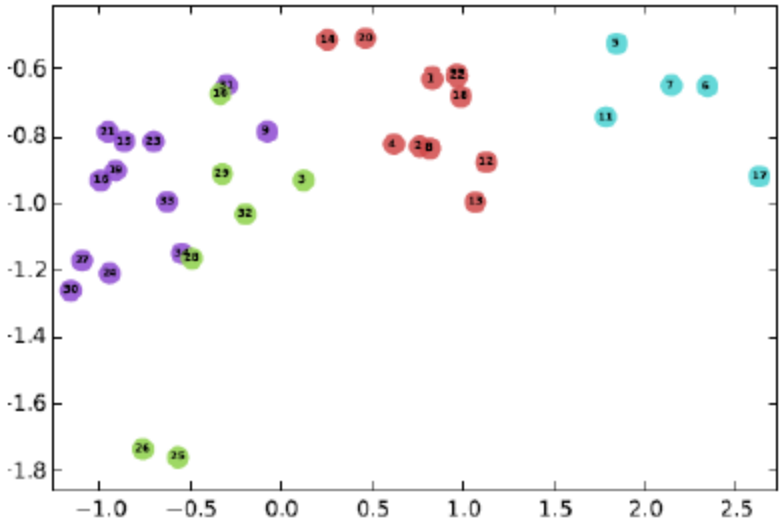


Example

- Zachary's Karate Network:



Input



Output

Problems with Previous Models

- Classical graph embedding algorithms
  - MDS, IsoMap, LLE, Laplacian Eigenmap
  - Most of them follow a matrix factorization or computation approach
  - **Hard to scale up**
  - **Difficult to extend to new settings**

Outline

- Preliminaries
  - word2vec
- Basic Network Embedding Models
  - DeepWalk, Node2vec, LINE, GrapRep, SDNE
- Advanced Network Embedding Models
  - Beyond embedding, vertex information, edge information
- Applications of Network Embedding
  - Basic applications, visualization, text classification, recommendation
- Conclusion

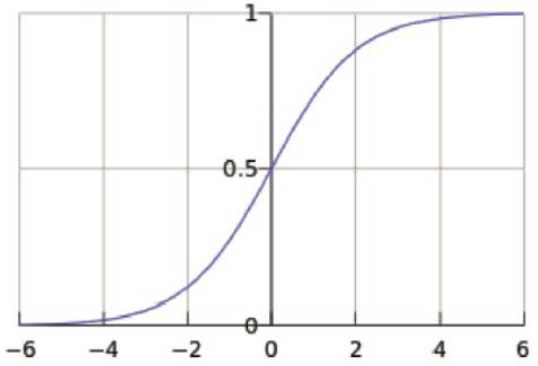
Preliminaries

- Softmax functions
- Distributional semantics
- Word2vec
  - CBOW
  - Skip-gram

Sigmoid Function

- A sigmoid function can map a real value to the interval (0, 1)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Softmax function

- It transforms a *K*-dimensional real vector into a *probability distribution*
  - A common transformation function to derive objective functions for classification or discrete variable modeling

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad \text{for } j = 1, \cdots, K.$$

Distributional semantics

- Target word = “stars”

he curtains open and the stars shining in on the barely
ars and the cold , close stars " . And neither of the w
rough the night with the stars shining so brightly , it
made in the light of the stars . It all boils down , wr
surely under the bright stars , thrilled by ice-white
sun , the seasons of the stars ? Home , alone , Jay pla
m is dazzling snow , the stars have risen full and cold
un and the temple of the stars , driving out of the hug
in the dark and now the stars rise , full and amber a
bird on the shape of the stars over the trees in front
But I could n’t see the stars or the moon , only the
they love the sun , the stars and the stars . None of
r the light of the shiny stars . The plash of flowing w
man ’s first look at the stars ; various exhibits , aer
rief information on both stars and constellations, inc

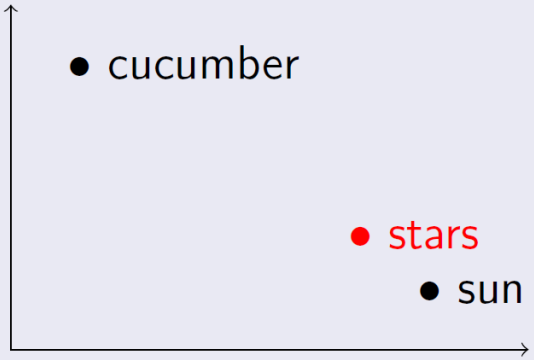
Distributional semantics

- Collect the contextual words for “stars”

Construct vector representations

	shining	bright	trees	dark	look
stars	38	45	2	27	12

Similarity in meaning as vector similarity



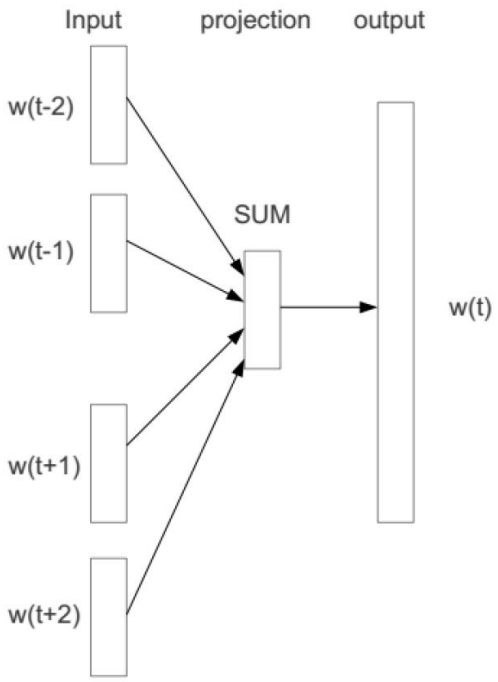
Word2Vec

- Input: a sequence of words from a vocabulary ***V***
- Output: a fixed-length vector for each term in the vocabulary
  - $\mathbf{v}_w$

It implements the idea of distributional semantics using a shallow neural network model.

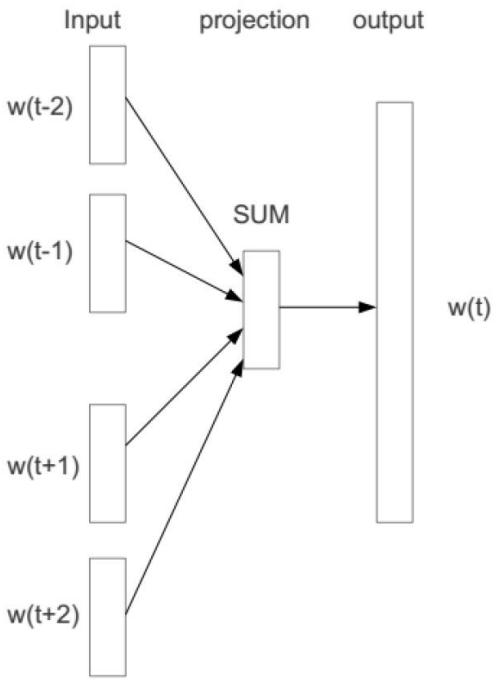
Architecture 1: CBOW

- **CBOW** predicts the current word using surrounding contexts
  - *Pr*(w_t | context(w_t))
    - Window size 2c
    - context(w_t) = [w_{t-c}, …, w_{t+c}]



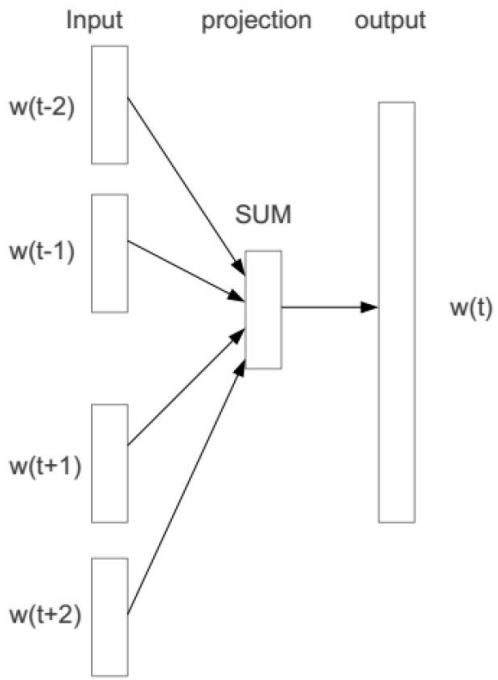
Architecture 1: CBOW

- **CBOW** predicts the current word using surrounding contexts
  - $Pr(w_t \mid \text{context}(w_t))$
  - Using a K-dimensional vector to represent words
    - $w_t \rightarrow \boldsymbol{v}_{w_t}$
    - $\tilde{\boldsymbol{v}}_{w_t} = \frac{\sum_{i=t-c}^{t+c} \boldsymbol{v}_{w_i}}{2c} \qquad (i \neq t)$



Architecture 1: CBOW

- **CBOW** predicts the current word using surrounding contexts
  - $Pr(w_t \mid \text{context}(w_t))$
  - Basic Idea
    - Given the context of the current word $\tilde{\boldsymbol{v}}_{w_t}$
    - Sim($\tilde{\boldsymbol{v}}_{w_t}$, $\boldsymbol{v}_{w_t}$) > Sim($\tilde{\boldsymbol{v}}_{w_t}$, $\boldsymbol{v}_{w_j}$)



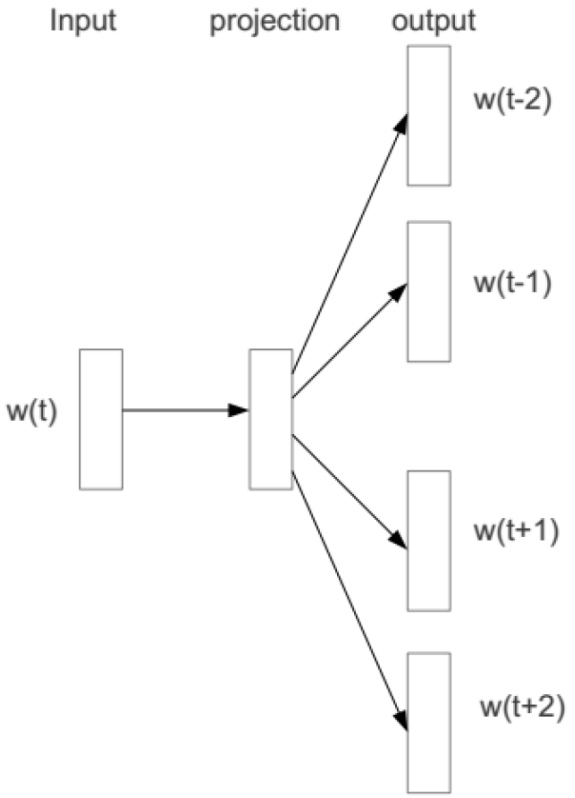
Architecture 1: CBOW

- How to formulate the idea
  - Using a softmax function
  - Considered as a classification problem
    - Each word is a classification label

$$P(w|w_{\text{context}}) = \frac{\exp(sim(\tilde{\boldsymbol{v}}_w, \boldsymbol{v}_w))}{\sum_{w'} \exp(sim(\tilde{\boldsymbol{v}}_w, \boldsymbol{v}_{w'}))}$$

Architecture 2

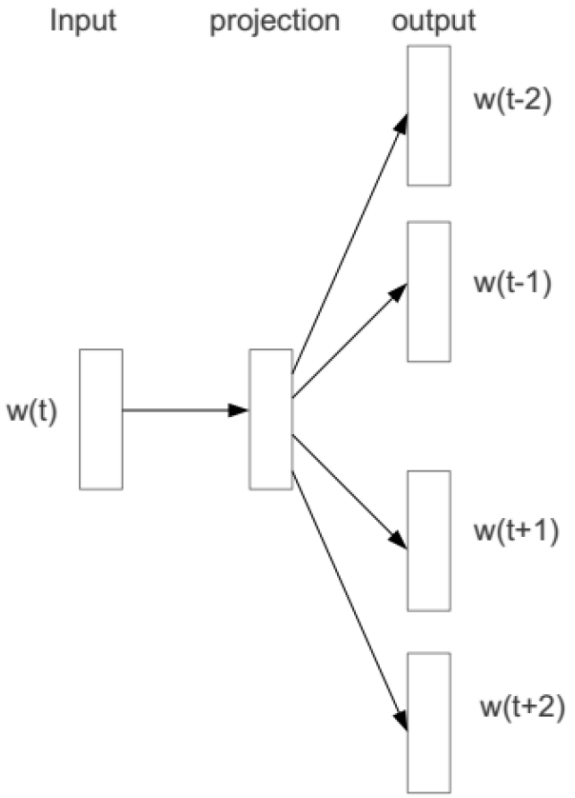
- **Skip-gram** predicts surrounding words using the current word
  - *Pr*($\text{context}(w_t) \mid w_t$)
    - Window size 2c
    - $\text{context}(w_t)$ = [w_{t-c}, …, w_{t+c}]



Architecture 2

- **Skip-gram** predicts surrounding words using the current word
  - *Pr*(context(w_t) | w_t)
    - Window size 2c
    - context(w_t) = [w_{t-c}, …, w_{t+c}]

$$P(w'|w) = \frac{\exp(sim(\boldsymbol{v}_w, \boldsymbol{v}_{w'}))}{\sum_{w''} \exp(sim(\boldsymbol{v}_w, \boldsymbol{v}_{w''}))}$$



Time Complexity for Direct Optimization

CBOW

$$P(w|w_{\text{context}}) = \frac{\exp(sim(\tilde{\boldsymbol{v}}_w , \boldsymbol{v}_w))}{\sum_{w\prime} \exp(sim(\tilde{\boldsymbol{v}}_w , \boldsymbol{v}_{w\prime}))}$$

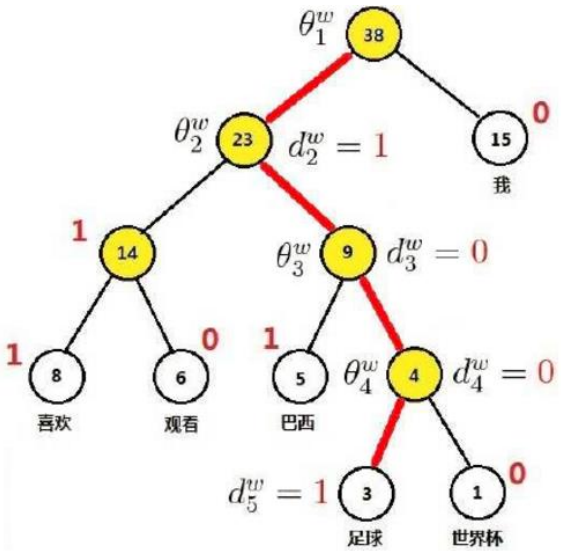
Skip-Gram

$$P(w'|w) = \frac{\exp(sim(\boldsymbol{v}_w , \boldsymbol{v}_{w\prime}))}{\sum_{w\prime\prime} \exp(sim(\boldsymbol{v}_w , \boldsymbol{v}_{w\prime\prime}))}$$

$O(|V|)$

Optimization I: Hierarchical Softmax

O($\log_2$ |V|)



It can be verified that the sum of the probabilities for all the leaf nodes is equal to 1

Optimization II: Negative Sampling

O(1+*k*)

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

Network Embedding Models

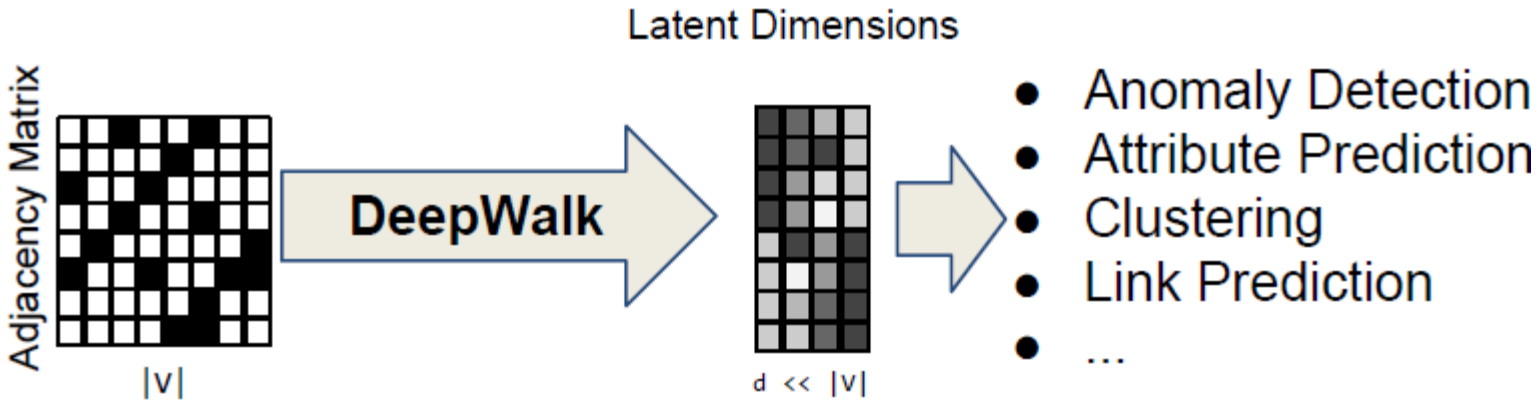
- Embedding
  - Neighborhood
    - DeepWalk
    - Node2vec
  - Proximity
    - LINE
    - GraRep
- Deep+promixity
  - SDNE

Network Embedding Models

- **DeepWalk** (Perozzi et al., KDD 2014)
- Node2vec
- LINE
- GraRep
- SDNE

DeepWalk

- DeepWalk learns a latent representation of adjacency matrices using deep learning techniques developed for language modeling



Language modeling

- Learning a representation of a word from documents (word co-occurrence):
  - word2vec: $\Phi : v \in V \mapsto \mathbb{R}^{|V| \times d}$
- The learned representations capture inherent structure
- Example:

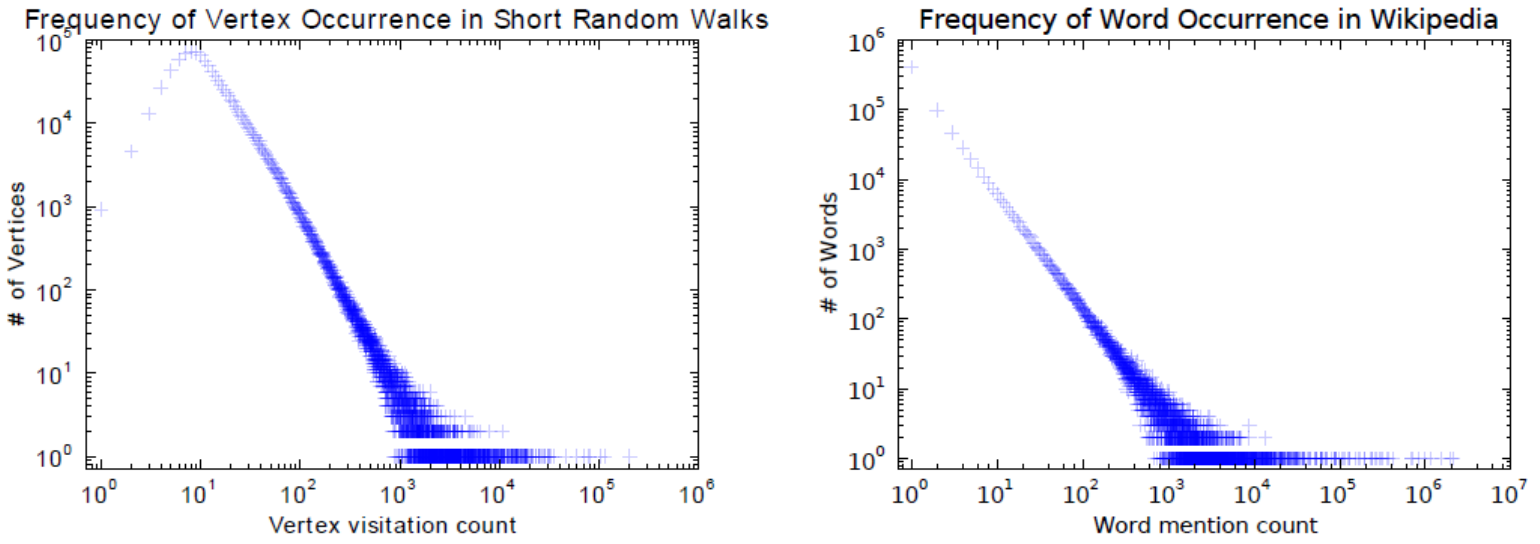
$$||\Phi(rose) - \Phi(daisy)|| < ||\Phi(rose) - \Phi(tiger)||$$

From language modeling to graphs

- Idea:
  - Nodes <--> Words
  - Node sequences <--> Sentences
- Generating node sequences:
  - Using random walks
    - short random walks = sentences

- Connection:
  - **Words frequency** in a natural language corpus follows a power law.
  - **Vertex frequency** in random walks on scale free graphs also follows a power law.

Power-law distribution $f(x) = ax^{-k}$



(a) YouTube Social Graph (b) Wikipedia Article Text

Figure 2: The distribution of vertices appearing in short random walks (2a) follows a power-law, much like the distribution of words in natural language (2b).

Scale-free network

- A **scale-free** network is a network whose degree distribution follows a **power law**, at least asymptotically.

$$P(k) \sim k^{-\gamma}$$

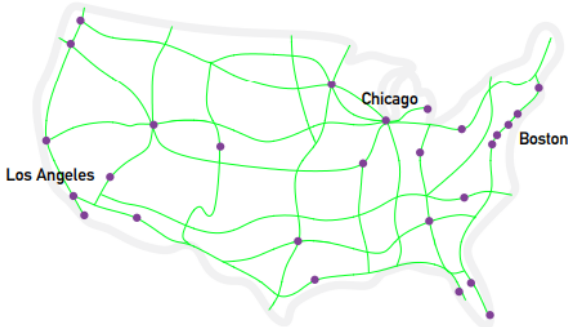
- Power-law distributions are scale invariance

Given a relation $f(x) = ax^{-k}$, scaling the argument x by a constant factor c causes only a proportionate scaling of the function itself.

$$f(cx) = a(cx)^{-k} = c^{-k} f(x) \propto f(x)$$

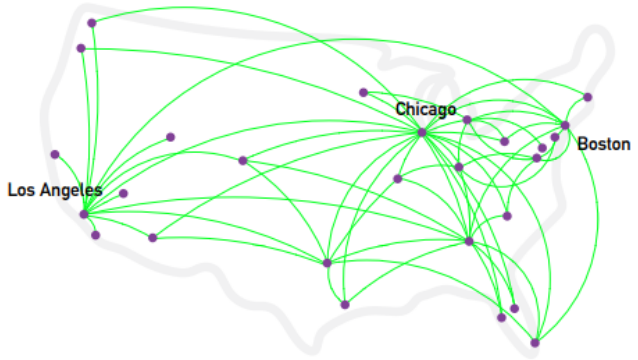
Scale-free network

- Which network is scale-free?



A random network looks a bit like the national highway network in which nodes are cities and links are the major highways. There are no cities with hundreds of highways and no city is disconnected from the highway system.

The degrees of a random network follow a Poisson distribution, rather similar to a bell curve. Therefore most nodes have comparable degrees and nodes with a large number of links are absent.

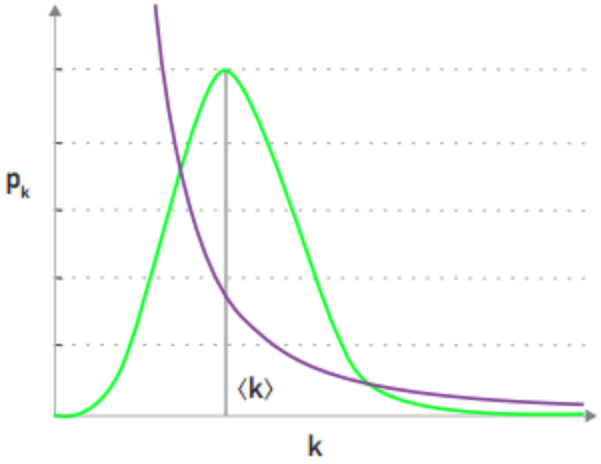


A scale-free network looks like the air-traffic network, whose nodes are airports and links are the direct flights between them. Most airports are tiny, with only a few flights. Yet, we have a few very large airports, like Chicago or Los Angeles, that act as major hubs, connecting many smaller airports.

In a network with a power-law degree distribution most nodes have only a few links. These numerous small nodes are held together by a few highly connected hubs.

Scale-free network

- Random networks' degree distribution follows a Poisson distribution, while scale-free network follows a power-law distribution



Random Network
Randomly chosen node: $k = \langle k \rangle \pm \langle k \rangle^{1/2}$
Scale: $\langle k \rangle$

Scale-Free Network
Randomly chosen node: $k = \langle k \rangle \pm \infty$
Scale: none

Lack of an Internal Scale

For any exponentially bounded distribution, like a Poisson or a Gaussian, the degree of a randomly chosen node is in the vicinity of $\langle k \rangle$. Hence $\langle k \rangle$ serves as the network's *scale*. For a power law distribution the second moment can diverge, and the degree of a randomly chosen node can be significantly different from $\langle k \rangle$. Hence $\langle k \rangle$ does not serve as an intrinsic scale. As a network with a power law degree distribution lacks an intrinsic scale

Note 1

- How to generate random paths based on a graph
  - Weighted graph
  - Unweighted graph

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

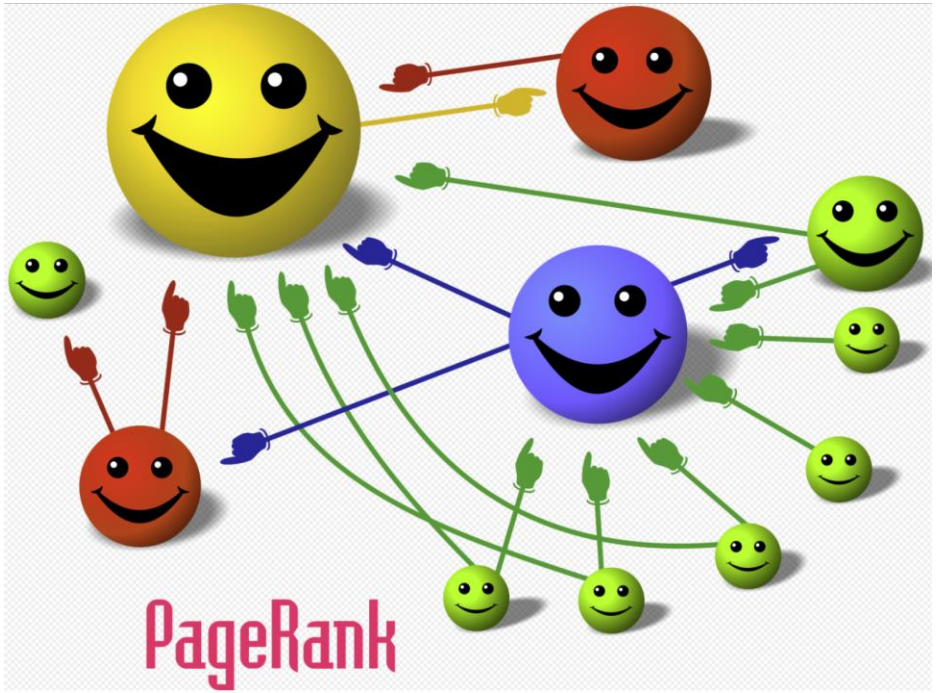
where π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant.

Note 2

- How to sample a discrete variable from a multinomial distribution

Note 3

- What is PageRank?



$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

The workflow of DeepWalk

2 Random Walks

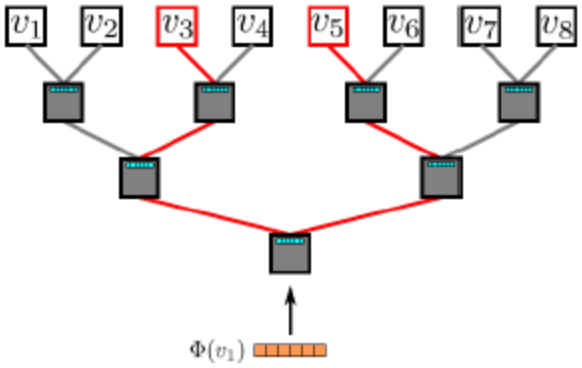
$\mathcal{W}_{v_4} = 4$

$u_k \begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix} v_j \rightarrow \Phi$ (d columns, j)

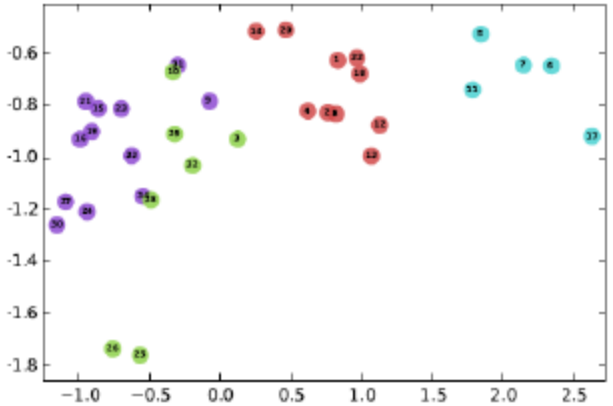
1
⋮

1 Input: Graph

3 Representation Mapping



4 Hierarchical Softmax



5 Output: Representation

Representation Mapping

$\mathcal{W}_{v_4} = v_4 \rightarrow v_3 \rightarrow v_1 \rightarrow v_5 \rightarrow v_1 \rightarrow v_{46} \rightarrow v_{51} \rightarrow v_{89}$

$\mathcal{W}_{v_4} = 4$

u_k [3, 1, 5] v_j → Φ (d × j)

1

⋮

- Map the vertex under focus (v_1) to its representation.
- Define a window of size w
- If w = 1 and $v = v_1$

Maximize: $\Pr(v_3|\Phi(v_1))$

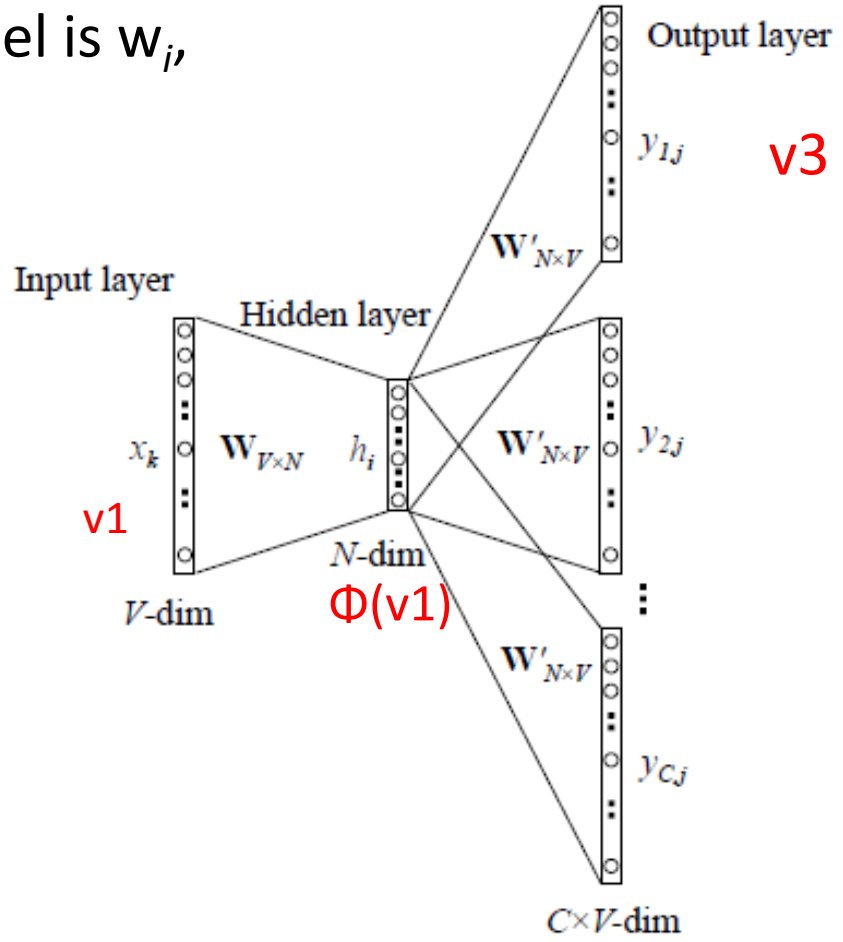
$\Pr(v_5|\Phi(v_1))$

Deep Learning Structure: Skip-gram model

Skip-gram: The input to the model is w_i, and the output could be $w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}$

Maximize: $\Pr(v_3|\Phi(v_1))$

$\Pr(v_5|\Phi(v_1))$



Experiments

- Node Classification
  - Some nodes have labels, some don’t
- DataSet
  - BlogCatalog
  - Flickr
  - YouTube

Results: BlogCatalog

	% Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1(%)	DeepWalk	**36.00**	**38.20**	**39.60**	**40.30**	**41.00**	**41.30**	41.50	41.50	42.00
	SpectralClustering	31.06	34.95	37.27	38.93	39.97	40.99	**41.66**	**42.42**	**42.62**
	EdgeCluster	27.94	30.76	31.85	32.99	34.12	35.00	34.63	35.99	36.29
	Modularity	27.35	30.74	31.77	32.97	34.09	36.13	36.08	37.23	38.18
	wvRN	19.51	24.34	25.62	28.82	30.37	31.81	32.19	33.33	34.28
	Majority	16.51	16.66	16.61	16.70	16.91	16.99	16.92	16.49	17.26
Macro-F1(%)	DeepWalk	**21.30**	**23.80**	25.30	26.30	27.30	27.60	27.90	28.20	28.90
	SpectralClustering	19.14	23.57	**25.97**	**27.46**	**28.31**	**29.46**	**30.13**	**31.38**	**31.78**
	EdgeCluster	16.16	19.16	20.48	22.00	23.00	23.64	23.82	24.61	24.92
	Modularity	17.36	20.00	20.80	21.85	22.65	23.41	23.89	24.20	24.97
	wvRN	6.25	10.13	11.64	14.24	15.86	17.18	17.98	18.86	19.57
	Majority	2.52	2.55	2.52	2.58	2.58	2.63	2.61	2.48	2.62

Network Embedding Models

- DeepWalk
- **Node2vec** (Grover et al., KDD 2016)
- GENE
- LINE
- SDNE

Node2Vec

- A generalized version of DeepWalk
  - Objective function

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u)).$$

  - Conditional independence

$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u)).$$

  - Symmetry in feature space

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}.$$

Node2Vec

$N_S(u) \subset V$

- A network neighborhood of node *u* generated through a neighborhood sampling strategy ***S***.
- The key lies in how to find a neighbor on the graph
- How does *DeepWalk* solve this?

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant.

How Node2vec Do this?

- Motivation

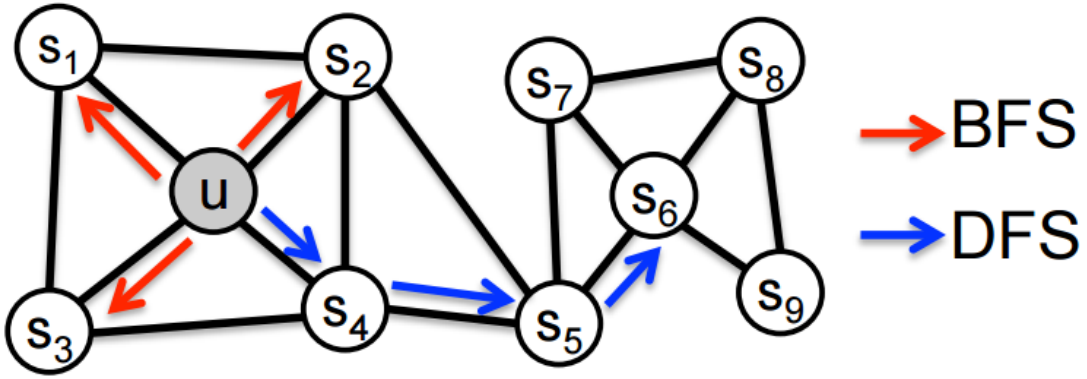


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

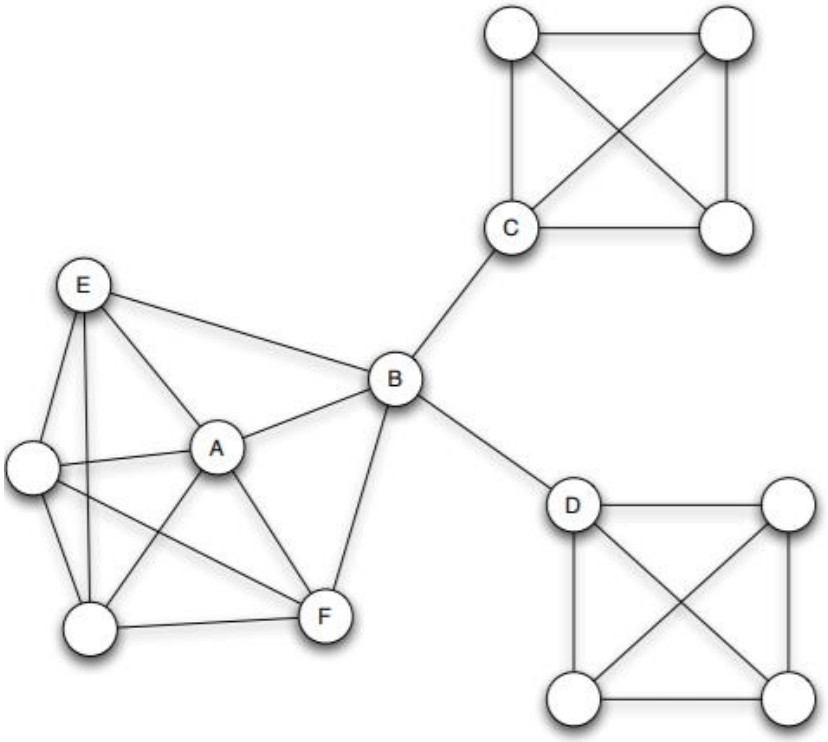
  - BFS: broader → ***homophily***
  - DFS: deeper → ***structural equivalence***

Homophily

- What is **Homophily**?
  - Homophily (i.e., "love of the same") is the tendency of individuals to associate and bond with similar others, as in the proverb "birds of a feather flock together".

Structural Holes

- Structural holes
  - The theory of structural holes [4] suggests that individuals would benefit from filling the “holes” (called as ***structural hole spanners***) between people or groups that are otherwise disconnected.
  - Just a representative kind of structural role



How Node2vec Do this?

- Can we combine the merits of DFS and BFS
  - BFS: broader → ***homophily***
  - DFS: deeper → ***structural equivalence***

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$$

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

How Node2vec Do this?

- Explaining the sampling strategy

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$$

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Return parameter, p**.** Parameter p controls the likelihood of immediately revisiting a node in the walk.

In-out parameter, q**.** Parameter q allows the search to differentiate between "inward" and "outward" nodes.

Node2vec Algorithm

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  $G' = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter = 1$ **to** r **do**
    for all nodes $u \in V$ **do**
      $walk$ = node2vecWalk(G', u, l)
      Append $walk$ to $walks$
  f = StochasticGradientDescent(k, d, $walks$)
  return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
  Inititalize $walk$ to $[u]$
  for $walk\_iter = 1$ **to** l **do**
    $curr = walk[-1]$
    V_{curr} = GetNeighbors($curr, G'$)
    s = AliasSample(V_{curr}, π)
    Append s to $walk$
  return $walk$

Alias Sampling

- In computing, the alias method is a family of efficient algorithms for sampling from a discrete probability distribution
- That is, it returns integer values $1 \leq i \leq n$ according to some arbitrary probability distribution pi.
- The algorithms typically use *O(n log n)* or *O(n)* preprocessing time, after which random values can be drawn from the distribution in *O(1)* time.

Comparison between DeepWalk and Node2vec

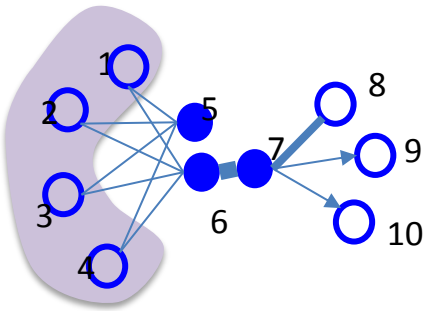
- Actually have the same objective function and formulations
- The difference lies in how to generate random walks
- BEAUTY: node → word, path → sentence

Network Embedding Models

- DeepWalk
- Node2vec
- **LINE** (Tang et al., WWW 2015)
- GraRep
- SDNE

LINE

First-order Proximity



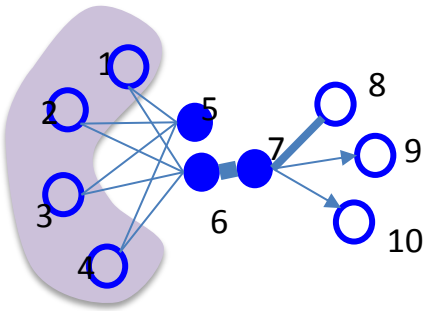
Vertex **6** and **7** have a large first-order proximity

- The local pairwise proximity between the vertices
  - Determined by the observed links
- However, many links between the vertices are missing
  - Not sufficient for preserving the entire network structure

From Jian Tang's slides

LINE

Second-order Proximity



Vertex **5** and **6** have a large second-order proximity

$\hat{p}_5 = (1,1,1,1,0,0,0,0,0,0)$

$\hat{p}_6 = (1,1,1,1,0,0,5,0,0,0)$

- The proximity between the *neighborhood structures* of the vertices
- Mathematically, the second-order proximity between each pair of vertices (u,v) is determined by:

$\hat{p}_u = (w_{u1}, w_{u2}, \ldots, w_{u|V|})$

$\hat{p}_v = (w_{v1}, w_{v2}, \ldots, w_{v|V|})$

From Jian Tang's slides

Questions

- How to characterize the first-order and second-order proximity?
  - We assume each node is associated with a low-dimensional latent factor

LINE

Preserving the First-order Proximity

- Given an ***undirected*** edge (v_i, v_j), the joint probability of v_i, v_j

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}$$

$\vec{u}_i$: Embedding of vertex v_i

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(i',j')} w_{i'j'}}$$

- Objective:

$$O_1 = d(\hat{p}_1(\cdot,\cdot), p_1(\cdot,\cdot))$$

KL-divergence

$$\propto -\sum_{(i,j)\in E} w_{ij} \log p_1(v_i, v_j)$$

From Jian Tang's slides

LINE

Preserving the Second-order Proximity

- Given a ***directed*** edge (v_i, v_j), the conditional probability of v_j given v_i is:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}$$

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in V} w_{ik}}$$

$\vec{u}_i$: Embedding of vertex *i* when *i* is a source node;
$\vec{u}_i'$: Embedding of vertex *i* when *i* is a target node.

- Objective:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i))$$

$$\propto -\sum_{(i,j) \in E} w_{ij} \log p_2(v_j|v_i)$$

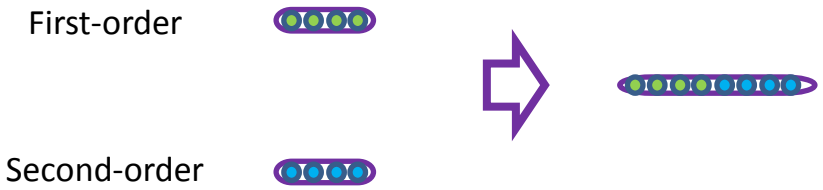
λ_i: Prestige of vertex in the network
$\lambda_i = \sum_j w_{ij}$

From Jian Tang's slides

LINE

Preserving both Proximity

- Concatenate the embeddings individually learned by the two proximity



From Jian Tang's slides

LINE

Optimization

- Stochastic gradient descent + Negative Sampling
  - Randomly sample an edge and multiple negative edges
- The gradient w.r.t the embedding with edge (i, j)

$$\frac{\partial O_2}{\partial \vec{u}_i} = w_{ij} \cdot \frac{\partial \log p_2(v_j|v_i)}{\partial \vec{u}_i}$$

Multiplied by the weight of the edge w_{ij}

- Problematic when the weights of the edges diverge
  - The scale of the gradients with different edges diverges
- Solution: **edge sampling**
  - Sample the edges according to their weights and treat the edges as binary
- Complexity: $O(dK|E|)$
  - Linear to the dimension d, the number of negative samples K, and the number of edges $|E|$

Questions

- How to model nodes with a small degree
- How to model new nodes

Solutions

Embedding Vertices of small degrees

- Sparse information in the neighborhood
- Solution: expand the neighbors by adding higher-order neighbors
  - e.g., neighbors of neighbors
  - breadth-first search
  - only consider the second-order neighbors

Embedding New Vertices

- Fix existing embeddings, and optimize w.r.t the new ones
- Objective

$$-\sum_{j \in N(i)} w_{ji} \log p_1(v_j, v_i) \quad \text{or}$$

$$-\sum_{j \in N(i)} w_{ji} \log p_2(v_j | v_i)$$

Network Embedding Models

- DeepWalk
- Node2vec
- **GraRep** (Cao et al., CIKM 2015)
- LINE
- SDNE

SG with NS ≈ Factorization on PMI matrix

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = PMI(w_i, c_j) - \log k$$

For a negative-sampling value of $k = 1$, the SGNS objective is factorizing a word-context matrix in which the association between a word and its context is measured by $f(w, c) = PMI(w, c)$. We refer to this matrix as the *PMI matrix*, M^{PMI}. For negative-sampling values $k > 1$, SGNS is factorizing a *shifted PMI matrix* $M^{PMI_k} = M^{PMI} - \log k$.

$$PMI(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

$$PMI(w, c) = \log \frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)}$$

Omer Levy, Yoav Goldberg:
Neural Word Embedding as Implicit Matrix Factorization. NIPS 2014: 2177-2185

Factorization with *k*-step context

$$
\begin{aligned}
& PMI(x,y) \\
= \; & \log \frac{P(x,y)}{P(x)P(y)} \\
= \; & \log \frac{P(y|x)}{P(y)} \\
= \; & \log \frac{P(y|x)}{\sum_{x'} P(y,x')} \\
= \; & \log \frac{P(y|x)}{\sum_{x'} P(y|x')P(x')}
\end{aligned}
$$

k-step probability transition matrix

$$A^k = \underbrace{A \cdots A}$$

$$p_k(c|w) = A^k_{w,c}$$

$$Y^k_{i,j} = W^k_i \cdot C^k_j = \log\left(\frac{A^k_{i,j}}{\sum_t A^k_{t,j}}\right) - \log(\beta)$$

DeepWalk as MF

- In SG with NS, we factorize the sum of *k*-step transition matrices

$$Y_{i,j}^{E-SGNS} = \log\left(\frac{M_{i,j}}{\sum_t M_{t,j}}\right) - \log(\beta)$$

$$M = A^1 + A^2 + \cdots + A^K$$

Cheng Yang, Zhiyuan Liu: Comprehend DeepWalk as Matrix Factorization. CoRR abs/1501.00358 (2015)

Problems with DeepWalk

- Neighbors are not equal in the random walks

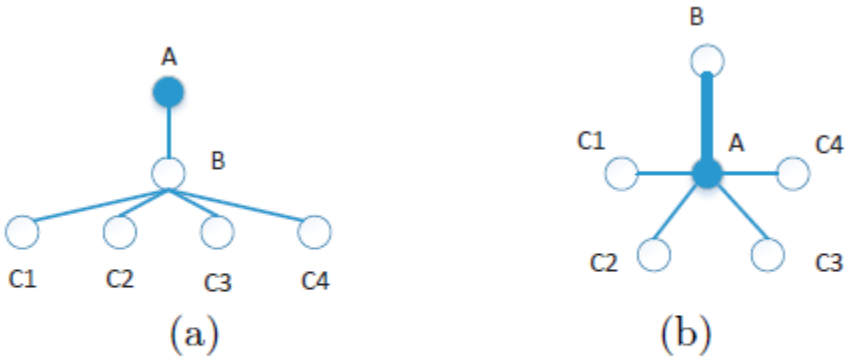


Figure 2: The importance of maintaining different k-step information separately in the graph representations.

Use all *k*-step representations

Table 1: Overall Algorithm

GraRep Algorithm

Input

Adjacency matrix S on graph
Maximum transition step K
Log shifted factor β
Dimension of representation vector d

1. Get k-step transition probability matrix A^k

Compute $A = D^{-1}S$
Calculate $A^1, A^2, \ldots, A^K$, respectively

2. Get each k-step representations

For $k = 1$ to K

2.1 Get positive log probability matrix
calculate $\Gamma_1^k, \Gamma_2^k, \ldots, \Gamma_N^k$ $(\Gamma_j^k = \sum_p A_{p,j}^k)$ respectively
calculate $\{X_{i,j}^k\}$

$$X_{i,j}^k = \log\left(\frac{A_{i,j}^k}{\Gamma_j^k}\right) - \log(\beta)$$

assign negative entries of X^k to 0

2.2 Construct the representation vector W^k
$[U^k, \Sigma^k, (V^k)^T] = SVD(X^k)$
$W^k = U_d^k (\Sigma_d^k)^{\frac{1}{2}}$

End for

3. Concatenate all the k-step representations

$W = [W^1, W^2, \ldots, W^K]$

Output

Matrix of the graph representation W

Experiments

Table 3: Results on 20-NewsGroup

Algorithm	200 samples			all data		
	3NG(200)	6NG(200)	9NG(200)	3NG(all)	6NG(all)	9NG(all)
GraRep	**81.12**	**67.53**	**59.43**	**81.44**	**71.54**	**60.38**
LINE (k-max=0)	80.36	64.88	51.58	80.58	68.35	52.30
LINE (k-max=200)	78.69	66.06	54.14	80.68	68.83	53.53
DeepWalk	65.58	63.66	48.86	65.67	68.38	49.19
DeepWalk (192dim)	60.89	59.89	47.16	59.93	65.68	48.61
E-SGNS	69.98	65.06	48.47	69.04	67.65	50.59
E-SGNS (192dim)	63.55	64.85	48.65	66.64	66.57	49.78
Spectral Clustering	49.04	51.02	46.92	62.41	59.32	51.91
Spectral Clustering (192dim)	28.44	27.80	36.05	44.47	36.98	47.36
Spectral Clustering (16dim)	69.91	60.54	47.39	78.12	68.78	57.87

Table 4: Results on Blogcatalog

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1	GraRep	**38.24**	**40.31**	**41.34**	**41.87**	**42.60**	**43.02**	**43.43**	**43.55**	**44.24**
	LINE	37.19	39.82	40.88	41.47	42.19	42.72	43.15	43.36	43.88
	DeepWalk	35.93	38.38	39.50	40.39	40.79	41.28	41.60	41.93	42.17
	E-SGNS	35.71	38.34	39.64	40.39	41.23	41.66	42.01	42.16	42.25
	Spectral Clustering	37.16	39.45	40.22	40.87	41.27	41.50	41.48	41.62	42.12
Macro-F1	GraRep	**23.20**	**25.55**	**26.69**	**27.53**	**28.35**	**28.78**	**29.67**	**29.96**	**30.93**
	LINE	19.63	23.04	24.52	25.70	26.65	27.26	27.94	28.68	29.38
	DeepWalk	21.02	23.81	25.39	26.27	26.85	27.36	27.67	27.96	28.41
	E-SGNS	21.01	24.09	25.61	26.59	27.64	28.08	28.33	28.34	29.26
	Spectral Clustering	19.26	22.24	23.51	24.33	24.83	25.19	25.36	25.52	26.21

Experiments

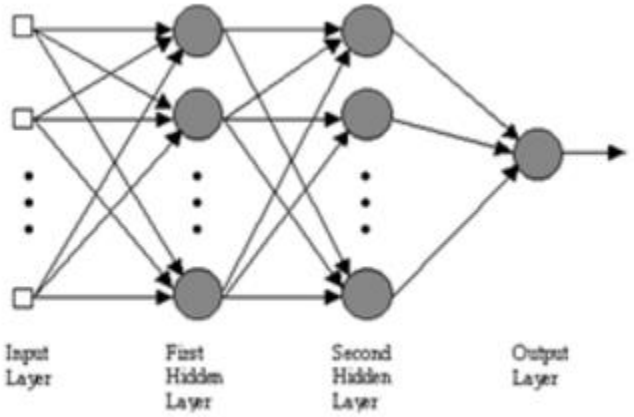
- Step length *k*
  - We found the performance of *K* = 4 is slightly better than *K*=3, while the results for *K*=5 is comparable to *K*=4.
  - We observed that the performance of *K*=7 is no better than that of *K*=6.

Network Embedding Models

- DeepWalk
- Node2vec
- GrapRep
- LINE
- **SDNE** (Wang et al., KDD 2016)

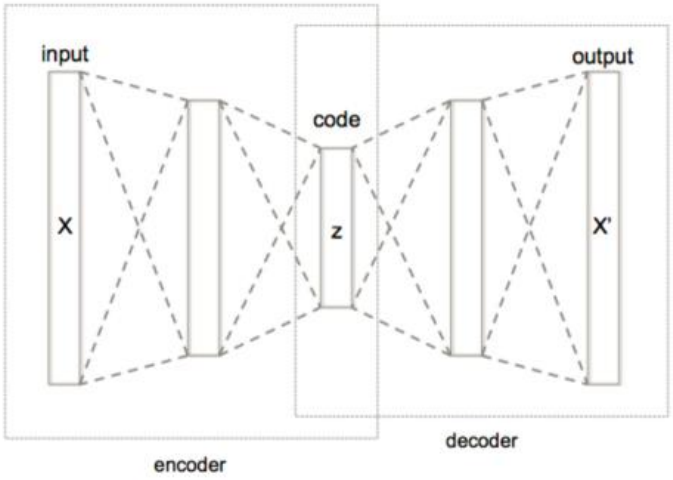
SDNE

- Preliminary
  - Multi-layer perceptron



SDNE

- Preliminary
  - Autoencoder



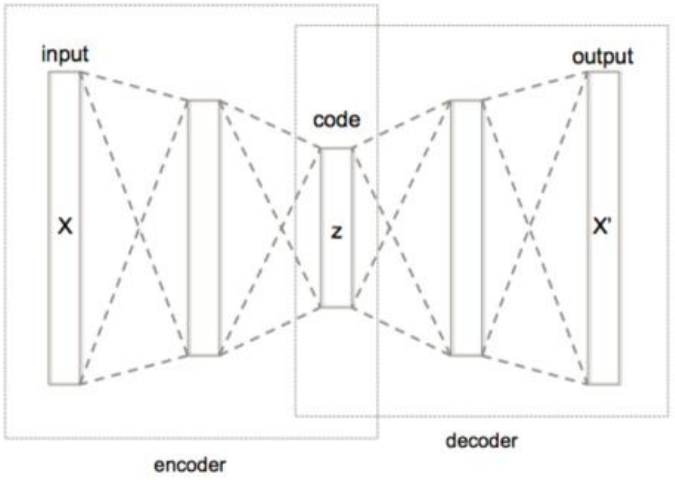
$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

$$\arg\min_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2$$

SDNE

- Preliminary
  - Autoencoder
    - The simplest case: a single hidden layer



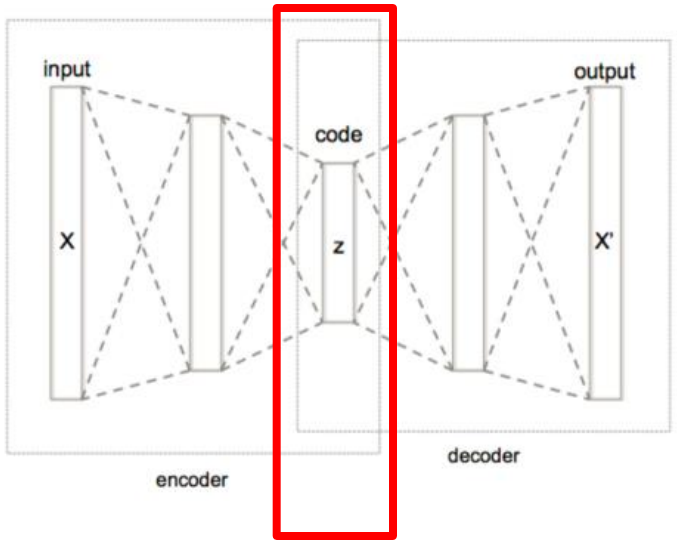
$$\mathbf{z} = \sigma_1(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x}' = \sigma_2(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2$$

SDNE

- Preliminary
  - Autoencoder
    - The simplest case: a single hidden layer



$$\mathbf{z} = \sigma_1(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x}' = \sigma_2(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

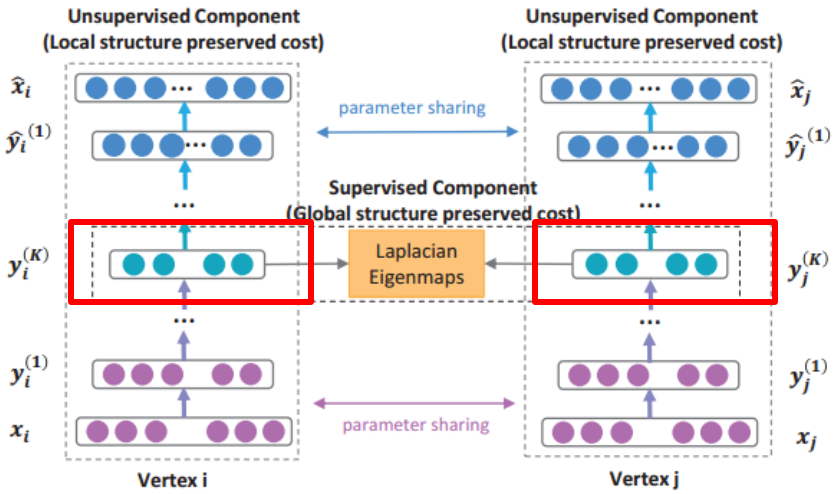
$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2$$

SDNE

- First-order proximity
  - Linked nodes should be coded similarly

$$\mathcal{L}_{1st} = \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2$$

$$= \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2$$



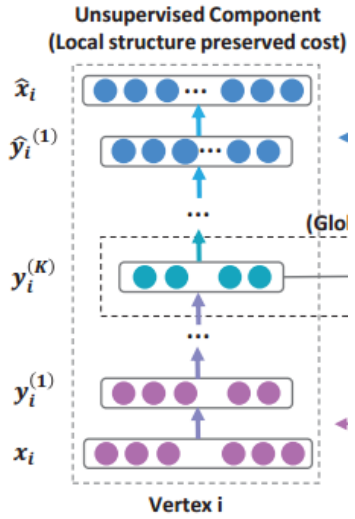
$$\mathbf{y}_i^{(1)} = \sigma(W^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)})$$

$$\mathbf{y}_i^{(k)} = \sigma(W^{(k)}\mathbf{y}_i^{(k-1)} + \mathbf{b}^{(k)}), k = 2, ..., K$$

SDNE

- Second-order proximity
  - The model should reconstruct the neighborhood vectors
  - Similar nodes even without links can have similar codes
    - Or we can not reconstruct the neighborhood

$$\mathcal{L}_{2nd} = \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b}_i\|_2^2$$
$$= \|(\hat{X} - X) \odot B\|_F^2$$



$$\mathbf{y}_i^{(1)} = \sigma(W^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)})$$
$$\mathbf{y}_i^{(k)} = \sigma(W^{(k)}\mathbf{y}_i^{(k-1)} + \mathbf{b}^{(k)}), k = 2, ..., K$$

SDNE

- Network reconstruction

Table 4: MAP on ARXIV-GRQC and BLOGCATALOG on reconstruction task

Method	ARXIV-GRQC					BLOGCATALOG				
	SDNE	*GraRep*	*LINE*	*DeepWalk*	*LE*	*SDNE*	*GraRep*	*LINE*	*DeepWalk*	*LE*
MAP	**0.836****	0.05	0.69	0.58	0.23	**0.63****	0.42	0.58	0.28	0.12

Significantly outperforms GraRep at the: ** 0.01 level.

- Link prediction

Table 5: *precision@k* on ARXIV GR-QC for link prediction

Algorithm	*P*@2	*P*@10	*P*@100	*P*@200	*P*@300	*P*@500	*P*@800	*P*@1000	*P*@10000
SDNE	**1**	**1**	**1**	**1**	**1***	**0.99****	**0.97****	**0.91****	**0.257****
LINE	1	1	1	1	0.99	0.936	0.74	0.79	0.2196
DeepWalk	1	0.8	0.6	0.555	0.443	0.346	0.2988	0.293	0.1591
GraRep	1	0.2	0.04	0.035	0.033	0.038	0.035	0.035	0.019
Common Neighbor	1	1	1	0.96	0.9667	0.98	0.8775	0.798	0.192
LE	1	1	0.93	0.855	0.827	0.66	0.468	0.391	0.05

Significantly outperforms Line at the: ** 0.01 and * 0.05 level, paired t-test.

Network Embedding Models

- DeepWalk
  - Node sentences + word2vec
- Node2vec
  - DeepWalk + more sampling strategies
- GrapRep
  - Separate MF with each *k*-step transition matrix
- LINE
  - Shallow + first-order + second-order proximity
- SDNE
  - Deep + First-order + second-order proximity

Advanced Models

- Beyond shallow embedding
- Incorporating vertex information
- Incorporating edge information

Beyond shallow embedding

- RNN base models **(Li et al., WWW 2017)**
  - Any sequence models can be applied to characterize node and **sequence representations**

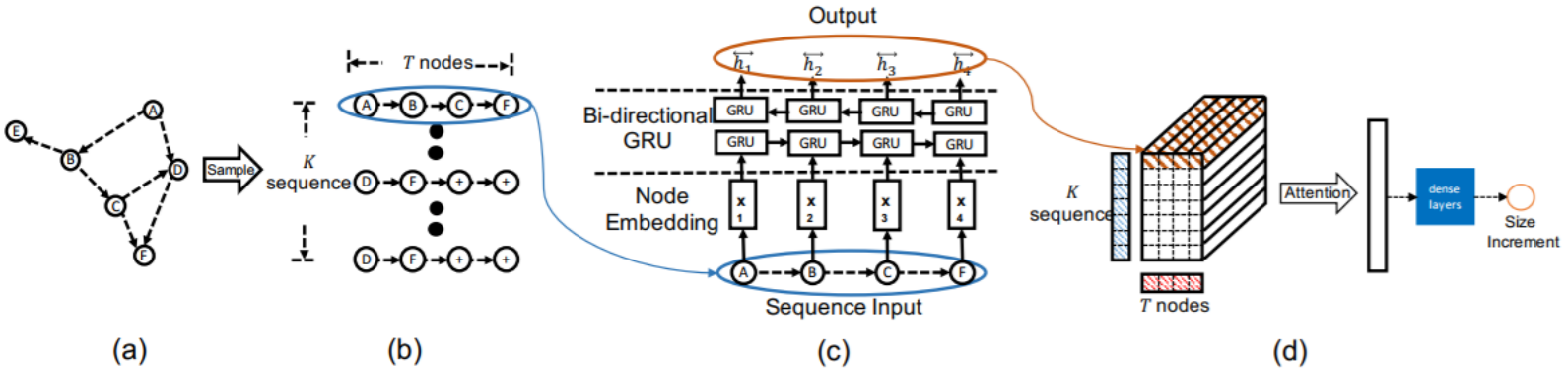
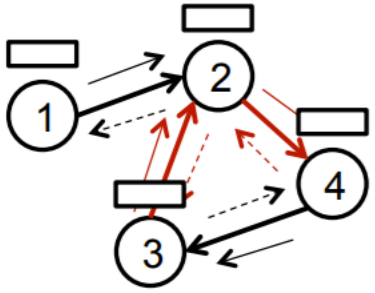


Figure 1: The end-to-end pipeline of DeepCas.

Beyond shallow embedding

- (Gated) Graph Neural Networks **(Li et al., 2015)**



Node representation for node *v* at propagation step *t*: $\mathbf{h}_v^{(t)}$

Propagate representations along edges, allow multiple edge types and propagation on both directions

Edge type and direction

$$\mathbf{h}_v^{(t)} = \sum_{v' \in \mathrm{IN}(v)} f(\mathbf{h}_{v'}^{(t-1)}, l_{(v',v)}) + \sum_{v' \in \mathrm{OUT}(v)} f(\mathbf{h}_{v'}^{(t-1)}, l_{(v,v')})$$

Example: $f(\mathbf{h}_{v'}^{(t-1)}, l_{(v',v)}) = \mathbf{A}^{(l_{(v,v')})}\mathbf{h}_{v'}^{(t-1)} + \mathbf{b}^{(l_{(v,v')})}$

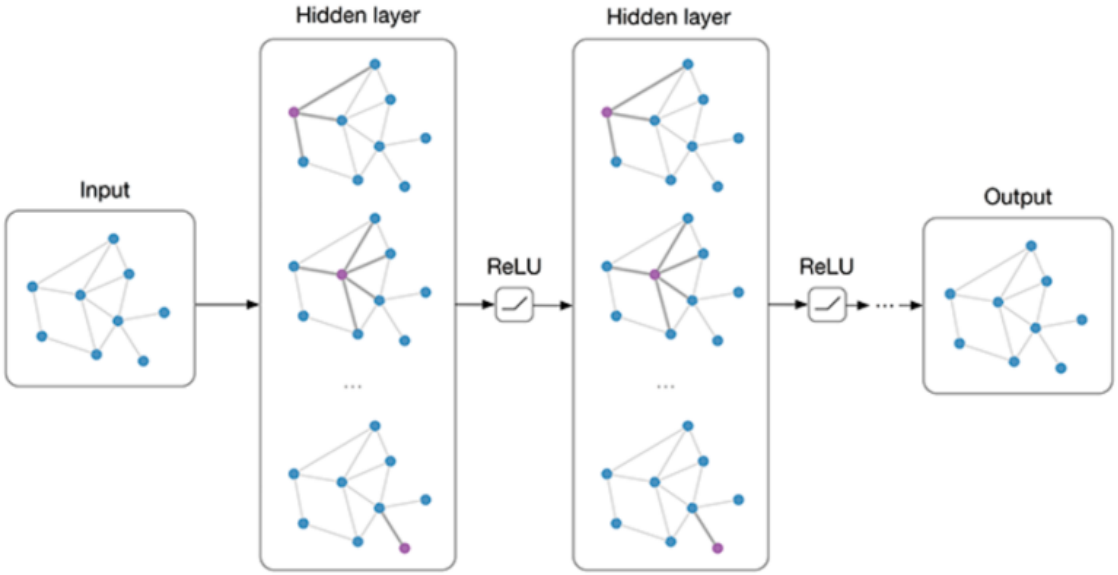
$$\mathbf{h}_v^{(1)} = [\mathbf{x}_v^\top, \mathbf{0}]^\top \qquad \mathbf{a}_v^{(t)} = \mathbf{A}_{v:}^\top [\mathbf{h}_1^{(t-1)\top} \cdots \mathbf{h}_{|\mathcal{V}|}^{(t-1)\top}]^\top$$

$$\mathbf{z}_v^{(t)} = \sigma(\mathbf{W}_z \mathbf{a}_v^{(t)} + \mathbf{U}\mathbf{h}_v^{(t-1)}) \qquad \mathbf{r}_v^{(t)} = \sigma(\mathbf{W}_r \mathbf{a}_v^{(t)} + \mathbf{U}_r \mathbf{h}_v^{(t-1)})$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}(\mathbf{r}_v^{(t)} \odot \mathbf{h}_v^{(t-1)})) \qquad \mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^{(t)}) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^{(t)} \odot \widetilde{\mathbf{h}_v^{(t)}}.$$

Beyond shallow embedding

- Graph Convolutional Networks **(Kipf et al., 2016)**



Multi-layer Graph Convolutional Network (GCN) with first-order filters.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) .$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$. $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$.

Incorporating Vertex Information

- Vertex labels **(Tu et al., IJCAI 2016)**
  - Given a ... el l_v
  - Goal: le ... abel
  - Solutio

$$\min_{X,Y,W,\xi} \mathcal{L} = \min_{X,Y,W,\xi} \mathcal{L}_{DW} + \frac{1}{2}\|W\|_2^2 + C\sum_{i=1}^{T}\xi_i$$

$$\text{s.t.} \quad \mathbf{w}_{l_i}^T\mathbf{x}_i - \mathbf{w}_j^T\mathbf{x}_i \geq e_i^j - \xi_i, \quad \forall i, j$$

  - (1) DeepWalk as MF

$$\min_{X,Y} \mathcal{L}_{DW} = \min_{X,Y} ||M - (X^TY)||_2^2 + \frac{\lambda}{2}(||X||_2^2 + ||Y||_2^2)$$

  - (2) Max-margin classifier

$$\min_{W,\xi} \mathcal{L}_{SVM} = \min_{W,\xi} \frac{1}{2}\|W\|_2^2 + C\sum_{i=1}^{T}\xi_i$$

$$\text{s.t.} \quad \mathbf{w}_{l_i}^T\mathbf{x}_i - \mathbf{w}_j^T\mathbf{x}_i \geq e_i^j - \xi_i, \quad \forall i, j$$

where

$$e_i^j = \begin{cases} 1, & \text{if} \quad l_i \neq j, \\ 0, & \text{if} \quad l_i = j. \end{cases}$$

Incorporating Edge Information

- Edges are with label information **(Tu et al., IJCAI 2017)**

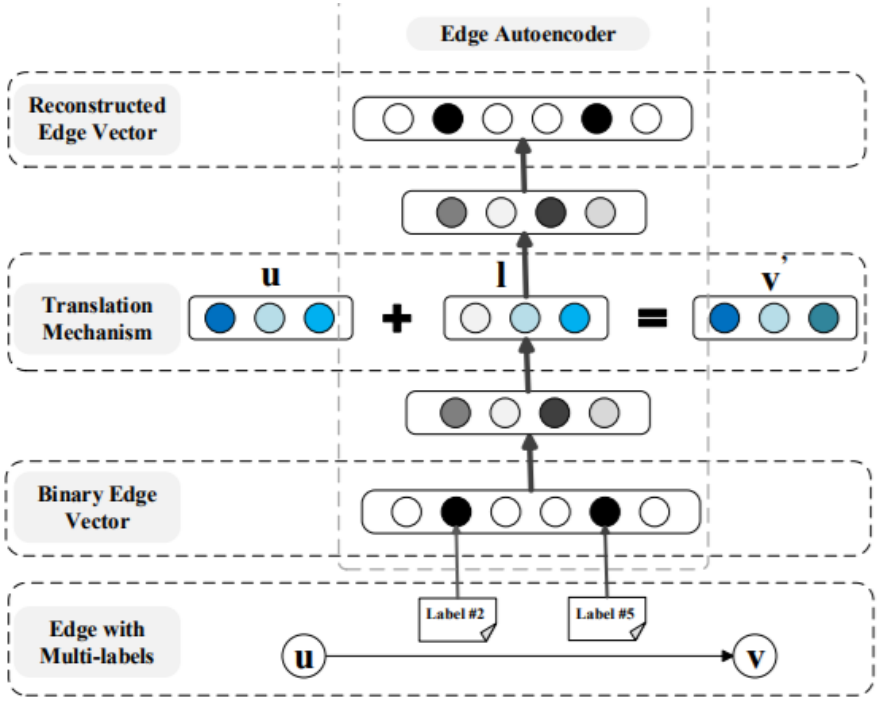


Figure 1: The framework of TransNet.

For each edge (u,v), we have

$$\mathbf{u} + \mathbf{l} \approx \mathbf{v}'.$$

TransE construction for networks with labeled edges

Applications of Network Embedding

- Basic applications
- Data Visualization
- Text classification
- Recommendation

Basic Applications

- Network reconstruction
- Link prediction
- Clustering
- Feature coding
  - Node classification
    - Demographic prediction
    - **Question: how to infer the demographics of a microblog user**

Applications of Network Embedding

- Basic applications
- **Data Visualization** (Tang et al., WWW 2016)
- Text classification
- Recommendation

Data Visualization

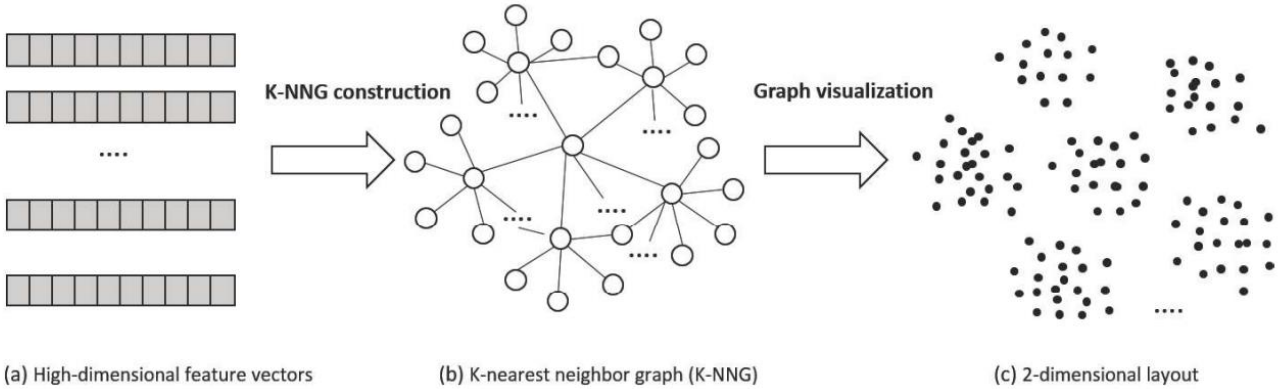


Figure 1: A typical pipeline of data visualization by first constructing a K-nearest neighbor graph and then projecting the graph into a low-dimensional space.

Data Visualization

- Construction of the KNN graph

For the weights of the edges in the K-nearest neighbor graph, we use the same approach as t-SNE. The conditional probability from data $\vec{x}_i$ to $\vec{x}_j$ is first calculated as:

$$p_{j|i} = \frac{\exp(-||\vec{x}_i - \vec{x}_j||^2/2\sigma_i^2)}{\sum_{(i,k)\in E}\exp(-||\vec{x}_i - \vec{x}_k||^2/2\sigma_i^2)}, \text{ and} \quad (1)$$
$$p_{i|i} = 0,$$

Then the graph is symmetrized through setting the weight between $\vec{x}_i$ and $\vec{x}_j$ as:

$$w_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \quad (2)$$

Data Visualization

- Visualization-based embedding

$$P(e_{ij} = 1) = f(||\vec{y}_i - \vec{y}_j||),$$

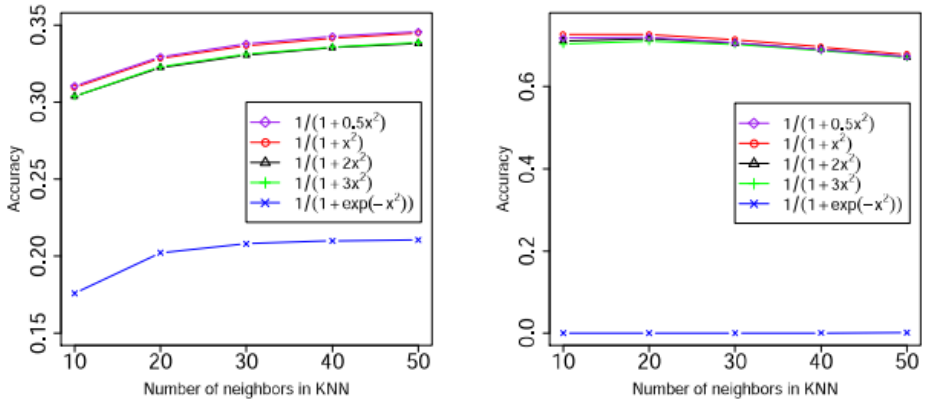
$$P(e_{ij} = w_{ij}) = P(e_{ij} = 1)^{w_{ij}}.$$

$$\begin{aligned} O &= \prod_{(i,j)\in E} p(e_{ij} = 1)^{w_{ij}} \prod_{(i,j)\in \bar{E}} (1 - p(e_{ij} = 1))^{\gamma} \\ &\propto \sum_{(i,j)\in E} w_{ij} \log p(e_{ij} = 1) + \sum_{(i,j)\in \bar{E}} \gamma \log(1 - p(e_{ij} = 1)), \end{aligned}$$

Data Visualization

- Non-linear function

$$P(e_{ij} = 1) = f(||\vec{y}_i - \vec{y}_j||),$$



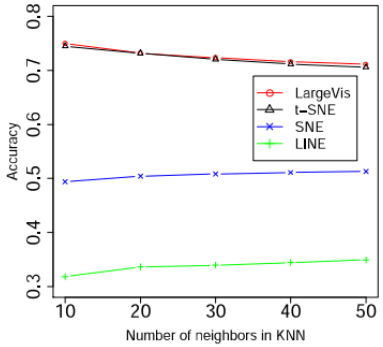
(a) WikiDoc (b) LiveJournal

Figure 4: Comparing different probabilistic functions.

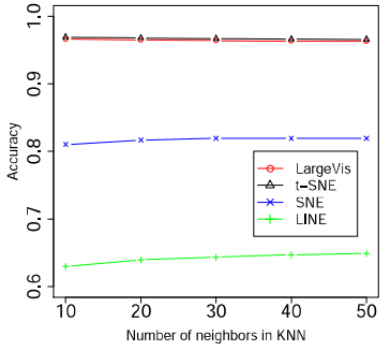
$$f(x) = \frac{1}{1+x^2}$$

Data Visualization

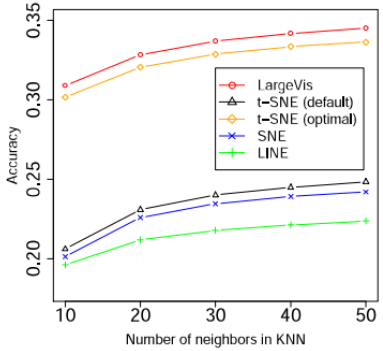
- Accuracy



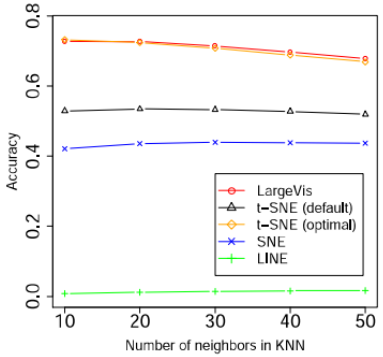
(a) 20NG



(b) MNIST



(c) WikiDoc



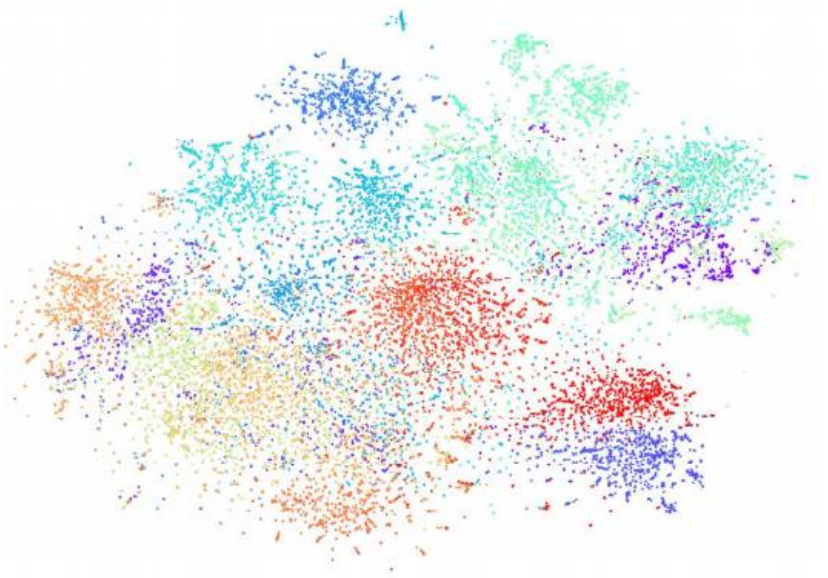
(d) LiveJournal

- Running time

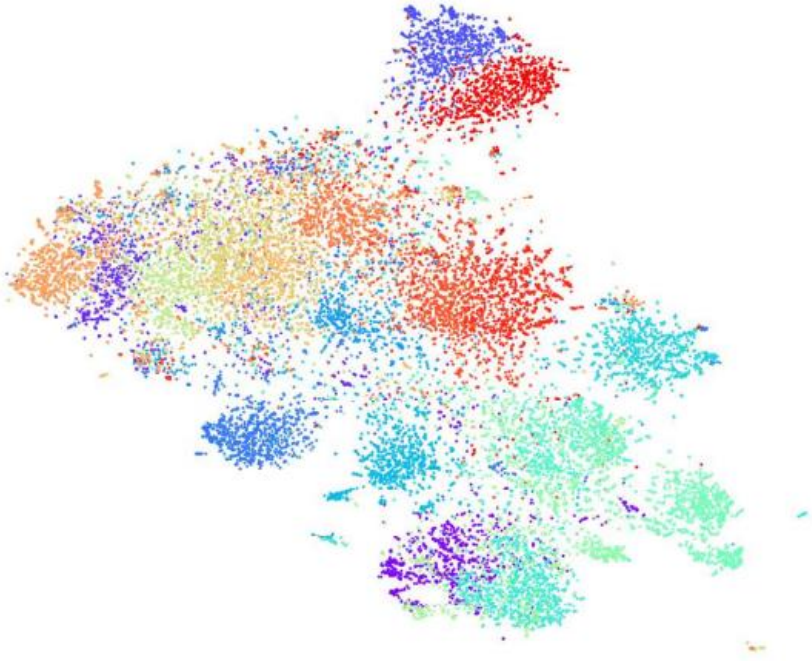
Table 2: Comparison of running time (hours) in graph visualization between the t-SNE and LargeVis.

Algorithm	20NG	MNIST	WikiWord	WikiDoc	LiveJournal	CSAuthor	DBLPPaper
t-SNE	0.12	0.41	9.82	45.01	70.35	28.33	18.73
LargeVis	0.14	0.23	2.01	5.60	9.26	4.24	3.19
Speedup Rate	0	0.7	3.9	7	6.6	5.7	4.9

Data Visualization



(a) 20NG (t-SNE)



(b) 20NG (LargeVis)

Applications of Network Embedding

- Basic applications
- Data Visualization
- **Text classification** (Tang et al., KDD 2015)
- Recommendation

Text Classification

Network embedding helps text modeling

Text representation, e.g., word and document representation, …

Deep learning has been attracting increasing attention …

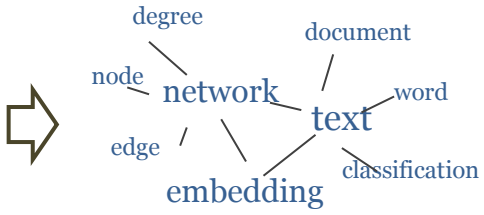
A future direction of deep learning is to integrate unlabeled data …

…

The Skip-gram model is quite effective and efficient …

Information networks encode the relationships between the data objects …

Free text



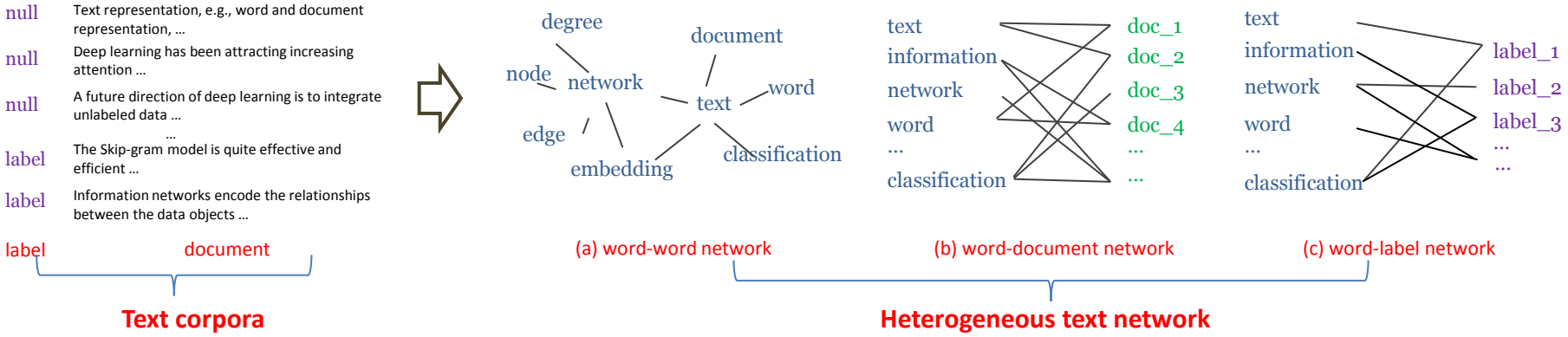
word co-occurrence network

If we have the word network, we can a network embedding model to learn word representations.

From Jian Tang's slides

Text Classification

- Adapt the advantages of unsupervised text embedding approaches but naturally utilize the ***labeled*** data for specific tasks
- Different levels of word co-occurrences*: local context-level, document-level, label-level*



From Jian Tang's slides

Text Classification

Bipartite Network Embedding

- Extend previous work **LINE** (Tang et al. WWW'2015) on large-scale information network embedding
  - Preserve the *first-order* and *second-order* proximity
  - Only consider the *second-order* proximity here

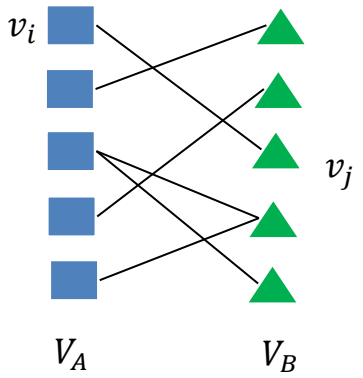
- For each edge (v_i, v_j), define a conditional probability

$$p(v_j|v_i) = \frac{\exp(\vec{u}_j^T \cdot \vec{u}_i)}{\sum_{j' \in B} \exp(\vec{u}_{j'}^T \cdot \vec{u}_i)}$$

- Objective:

$$O = -\sum_{(i,j) \in E} w_{ij} \log p(v_j|v_i)$$

- Edge sampling and negative sampling for optimization



Tang et al. **LINE: Large-scale Information Network Embedding.** WWW'2015

From Jian Tang's slides

Text Classification

Heterogeneous Text Network Embedding

- Heterogeneous text network: three bipartite networks
  - Word-word (word-context), word-document, word-label network
  - Jointly embed the three bipartite networks
- Objective

$$O_{pte} = O_{ww} + O_{wd} + O_{wl}$$

- where

$$O_{ww} = -\sum_{(i,j)\in E_{ww}} w_{ij} \log p(v_i|v_j)$$

Objective for **word-word** network

$$O_{wd} = -\sum_{(i,j)\in E_{wd}} w_{ij} \log p(v_i|d_j)$$

Objective for **word-document** network

$$O_{wl} = -\sum_{(i,j)\in E_{wl}} w_{ij} \log p(v_i|l_j)$$

Objective for **word-label** network

From Jian Tang's slides

Text Classification

Results on **Long** Documents: Predictive

		20newsgroup		Wikipedia		IMDB	
Type	Algorithm	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
Unsupervised	LINE(G_{wd})	79.73	78.40	80.14	80.13	89.14	89.14
Predictive embedding	CNN	78.85	78.29	79.72	79.77	86.15	86.15
	CNN(pretrain)	80.15	79.43	79.25	79.32	89.00	89.00
	PTE(G_{wl})	82.70	81.97	79.00	79.02	85.98	85.98
	PTE($G_{ww} + G_{wl}$)	83.90	83.11	81.65	81.62	89.14	89.14
	PTE($G_{wd} + G_{wl}$)	**84.39**	**83.64**	82.29	82.27	89.76	89.76
	PTE(pretrain)	82.86	82.12	79.18	79.21	86.28	86.28
	PTE(joint)	84.20	83.39	**82.51**	**82.49**	**89.80**	**89.80**

PTE(joint) > PTE(pretrain)

PTE(joint) > PTE(G_{wl})

PTE(joint) > CNN/CNN(pretrain)

From Jian Tang's slides

Text Classification

Results on **Short** Documents: Predictive

		DBLP		MR		Twitter	
Type	Algorithm	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
Unsupervised embedding	LINE ($G_{ww} + G_{wd}$)	74.22	70.12	71.13	71.12	73.84	73.84
Predictive embedding	CNN	76.16	73.08	72.71	72.69	**75.97**	**75.96**
	CNN(pretrain)	75.39	72.28	68.96	68.87	75.92	75.92
	PTE(G_{wl})	76.45	72.74	73.44	73.42	73.92	73.91
	PTE($G_{ww} + G_{wl}$)	76.80	73.28	72.93	72.92	74.93	74.92
	PTE($G_{wd} + G_{wl}$)	**77.46**	**74.03**	73.13	73.11	75.61	75.61
	PTE(pretrain)	76.53	72.94	73.27	73.24	73.79	73.79
	PTE(joint)	77.15	73.61	**73.58**	**73.57**	75.21	75.21

PTE(joint) > PTE(pretrain)

PTE(joint) > PTE(G_{wl})

PTE(joint) ≈ CNN/CNN(pretrain)

From Jian Tang's slides

Applications of Network Embedding

- Basic applications
- Data Visualization
- Text classification
- **Recommendation** (Zhao et al., AIRS 2016)

Traditional Recommendation Methods

- Popularity
- Item-KNN
- User-KNN
- Mixture of item-KNN and user-KNN
- Bayesian Personalized Ranking

Recommendation

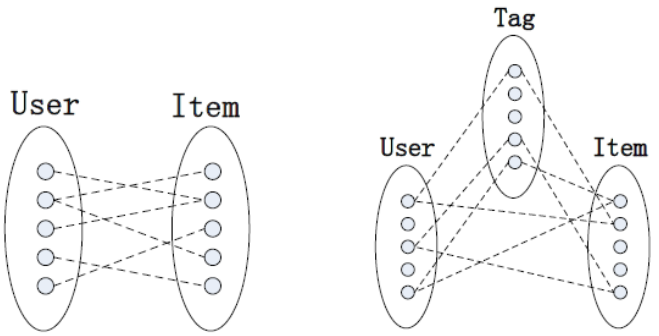
- From training records to networks

Definition 3. ***Bipartite User-Item (UI) Network***. *Let $\mathcal{U}$ denote the set of all the users, and $\mathcal{I}$ denote the set of all the items. A bipartite user-item network can be denoted by $\mathcal{G}^{(bi)} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, where the vertex set $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$, the edge set $\mathcal{E} \subset \mathcal{U} \times \mathcal{I}$, the weight matrix $\mathbf{W}$ stores the edge weights, and $W_{u,i}$ denote the link weight between a user u and an item i.*

Definition 4. ***Tripartite User-Item-Tag (UIT) Network***. *Let $\mathcal{U}$ denote the set of all the users, $\mathcal{I}$ denote the set of all the items, and $\mathcal{T}$ denote the set of all the tags. A tripartite user-item-tag network can be denoted by $\mathcal{G}^{(tri)} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, where the vertex set $\mathcal{V} = \mathcal{U} \cup \mathcal{I} \cup \mathcal{T}$, the edge set $\mathcal{E} \subset \big((\mathcal{U} \times \mathcal{I}) \cup (\mathcal{U} \times \mathcal{T}) \cup (\mathcal{I} \times \mathcal{T})\big)$, and the weight matrix $\mathbf{W}$ stores the edge weights.*

Recommendation

- Learning Distributed Representations for Recommender Systems with a Network Embedding Approach
  - Motivation



(a) User-item bipartite network.

(b) User-item-tag tripartite network.

Recommendation

- Given any edge in the network

$$P(e_s, e_t) = \sigma(\boldsymbol{v}_{e_s}^\top \cdot \boldsymbol{v}_{e_t}) = \frac{1}{1 + \exp(-\boldsymbol{v}_{e_s}^\top \cdot \boldsymbol{v}_{e_t})}.$$

$$\hat{P}(e_s, e_t) = \frac{W_{e_s,e_t}}{\sum_{(e_{s'},e_{t'})\in\mathcal{E}} W_{e_{s'},e_{t'}}}.$$

$$L(\mathcal{G}) = D_{KL}(\hat{P}(\cdot,\cdot)||P(\cdot,\cdot)) \propto \sum_{(e_s,e_t)\in\mathcal{E}} W_{e_s,e_t} \log P(e_s, e_t).$$

Recommendation

- User-item recommendation

Table 2. Performance comparisons of the proposed method and baselines on item recommendation.

Methods	JD				MovieLens			
	P@10	R@10	MAP	MRR	P@10	R@10	MAP	MRR
BPR	0.171	0.360	0.337	0.564	0.097	0.169	0.148	0.195
DeepWalk	0.259	0.443	0.502	0.806	0.203	0.243	0.249	0.358
NERM	**0.275**	**0.477**	**0.528**	**0.819**	**0.206**	**0.256**	**0.258**	**0.368**

Recommendation

- User-item-tag recommendation

Table 4. Performance comparisons of the proposed methods and baselines on tag recommendation.

Methods	Last.fm						Bookmarks					
	P@1	R@1	F@1	P@5	R@5	F@5	P@1	R@1	F@1	P@5	R@5	F@5
PITF	0.305	0.125	0.178	0.189	0.351	0.245	0.381	0.132	0.197	0.204	0.304	0.244
DeepWalk	0.088	0.044	0.059	0.040	0.099	0.057	0.064	0.024	0.035	0.038	0.074	0.050
NERM	**0.327**	**0.165**	**0.220**	0.182	**0.370**	0.244	**0.396**	**0.135**	**0.201**	**0.228**	**0.323**	**0.267**

Conclusions

- There are no boundaries between data types and research areas in terms of mythologies
  - Data models are the core
- Even if the ideas are similar, we can move from shallow to deep if the performance actually improves
- Task-specific and data-specific network embedding models are hot research topics

References

- Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013: 3111-3119
- Bryan Perozzi, Rami Al-Rfou', Steven Skiena. DeepWalk: online learning of social representations. KDD 2014: 701-710
- Aditya Grover, Jure Leskovec. node2vec: Scalable Feature Learning for Networks. KDD 2016: 855-864
- Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, Qiaozhu Mei. LINE: Large-scale Information Network Embedding. WWW 2015: 1067-1077
- Daixin Wang, Peng Cui, Wenwu Zhu. Structural Deep Network Embedding. KDD 2016: 1225-1234
- Jian Tang, Jingzhou Liu, Ming Zhang, Qiaozhu Mei. Visualizing Large-scale and High-dimensional Data. WWW 2016: 287-297
- Jian Tang, Meng Qu, Qiaozhu Mei. PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks. KDD 2015: 1165-1174
- Wayne Xin Zhao, Jin Huang, Ji-Rong Wen. Learning Distributed Representations for Recommender Systems with a Network Embedding Approach. AIRS 2016.
- Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, Maosong Sun. Max-Margin DeepWalk: Discriminative Learning of Network Representation. IJCAI 2016.
- Cunchao Tu, Zhengyan Zhang, Zhiyuan Liu, Maosong Sun. TransNet: Translation-Based Network Representation Learning for Social Relation Extraction. IJCAI 2017.
- Thomas N. Kipf, Max Welling: Semi-Supervised Classification with Graph Convolutional Networks. CoRR abs/1609.02907 (2016)
- Cheng Li, Jiaqi Ma, Xiaoxiao Guo, Qiaozhu Mei: DeepCas: An End-to-end Predictor of Information Cascades. WWW 2017: 577-586
- Yujia Li, Daniel Tarlow, Marc Brockschmidt, Richard S. Zemel: Gated Graph Sequence Neural Networks. CoRR abs/1511.05493 (2015)
- Shaosheng Cao, Wei Lu, Qiongkai Xu: GraRep: Learning Graph Representations with Global Structural Information. CIKM 2015: 891-900

Disclaimer

- For convenience, I directly copy some original slides or figures from the referred papers. I am sorry but I did not ask for the permission of each referred author. I thank you for these slides. I will not distribute your original slides.